PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/00, 3/00, G06T 13/00

A1

(11) International Publication Number: WO 00/36516

(43) International Publication Date: 22 June 2000 (22.06.00)

(21) International Application Number: PCT/US99/29961

(22) International Filing Date: 15 December 1999 (15.12.99)

(30) Priority Data:
60/112,232 15 December 1998 (15.12.98) US

(71) Applicant *(for all designated States except US)*: JAVU TECHNOLOGIES, INC. [US/US]; Chelsea Piers, Pier 62 – Suite M100, New York, NY 10011 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: ORLOV, Max E. [RU/US]; Apartment 22, 19 Jones Street, New York, NY 10014 (US). CHKLOVSKII, Dmitri D. [RU/US]; Apartment 1B, 333 East 49th Street, New York, NY 10017 (US). SAMOILOV, Michael [US/US]; Apartment 8, 7 Jones Street, New York, NY 10014 (US).

(74) Agents: LERCH, Joseph, B. et al.; Darby & Darby P.C., 805 Third Avenue, New York, NY 10022–7513 (US).

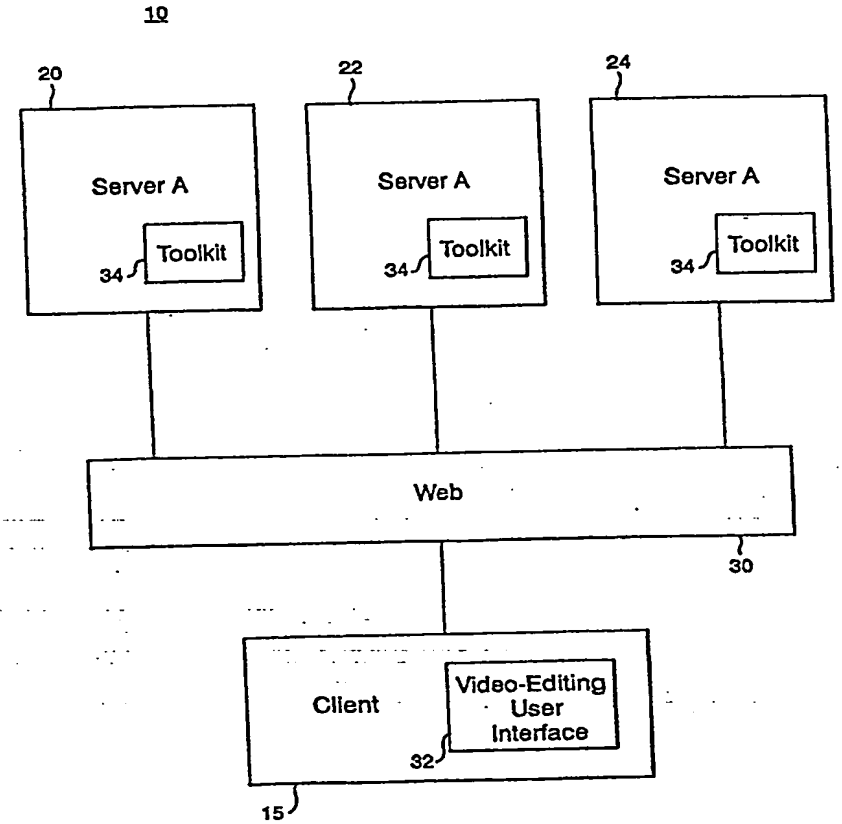
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: WEB–BASED VIDEO–EDITING METHOD AND SYSTEM

(57) Abstract

A network–based system (30) is provided for processing multimedia information. A server, preferably a group of servers (20, 22, 24), are coupled to the network and each incorporates a multimedia toolkit (34) or engine enabling creation, editing, viewing, and management of multimedia objects, such as files, including at least one of video, images, sound and animation. Each server includes storage for multimedia objects. A client (30), preferably a personal computer and preferably a plurality of clients, having access to the network each incorporate a multimedia–editing interface (32), preferably a graphic user interface (GUI), enabling the client to send multimedia processing commands through the network to the server from among a predefined set of multimedia processing commands. Preferably, a client can send a series of commands at a one time and each of several clients can send commands to be performed on the same object. The multimedia engine in each server acts on received multimedia processing commands from one or more clients by performing corresponding processing operations on multimedia objects previously stored by a client in the server's storage, and the server makes the processed multimedia objects available to clients over the network.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

WEB-BASED VIDEO-EDITING METHOD AND SYSTEM

STATEMENT OF GOVERNMENT INTEREST

This invention was partially funded by the Government under a grant from DARPA. The Government has certain rights in portions of the invention.

FIELD OF THE INVENTION

This invention relates generally to multimedia software and more particularly to libraries for use in building processing-intensive multimedia software for Web-based video-editing applications.

BACKGROUND OF THE INVENTION

The multimedia research community has traditionally focused its efforts on the compression, transport, storage and display of multimedia data. These technologies are fundamentally important for applications such as video conferencing and video-on-demand. The results of these efforts have made their way into many commercial products. For example, JPEG and MPEG, described below, are ubiquitous standards from image and audio/video compression. There are, however, problems in content-based retrieval and understanding, video production, and transcoding for heterogeneity and bandwidth adaptation. The lack of a high-performance toolkit that can be used to build processing-intensive multimedia applications is hindering development

in multimedia applications. In particular, in the area of video-editing, large volumes of data need to be stored, accessed and manipulated in an efficient manner. Solutions to the problems of storing video data include client-server applications and editing over the World Wide Web (Web). The existing multimedia toolkits, however, do not have sufficiently high performance to make these applications practical.

The data standards GIF, JPEG and MPEG dominate image and video data in the current state of the art. GIF (Graphics Interchange Format) is a bit-mapped graphics file format used commonly on the Web. JPEG (Joint Photographic Experts Group) is the internationally accepted standard for image data. JPEG is designed for compressing full color or gray-scale still images. For video data, including audio data, the international standard is MPEG (Moving Picture Experts Group). MPEG is actually a general reference to an evolving series of standards. For the sake of simplicity, the various MPEG versions will be referred to as the "MPEG standard" or simply "MPEG". The MPEG standard achieves a high rate of data compression by storing only the changes from one frame to another instead of an entire image.

The MPEG standard has four types of image coding for processing, the I-frame, the P-frame, the B-frame and the D-frame (from an early version of MPEG, but absent in later standards).

The I-frame (Intra-coded image) is self-contained, i.e. coded without any reference to other images. The I-frame is treated as a still image, and MPEG uses the JPEG standard to encode it. Compression in MPEG is often executed in real time and the compression rate of I-frames is the lowest within the MPEG standard. I-frames are used as points for random access in MPEG streams.

The P-frame (Predictive-coded frame) requires information of the previous I-frame in an MPEG stream, and/or all of the previous P-frames, for encoding and decoding. Coding of P-frames is based on the principle that areas of the image shift instead of change in successive images.

The B-frame (Bi-directionally predictive-coded frame) requires information from both the previous and the following I-frame and/or P-frame in the MPEG stream for encoding and decoding. B-frames have the highest compression ratio within the MPEG standard.

The D-frame (DC-coded frame) is intra-frame encoded. The D-frame is absent in more recent versions of the MPEG standard, however, applications are still required to deal with D-frames when working with the older MPEG versions. D-frames consist only of the lowest frequencies of an image. D-frames are used for display in fast-forward and fast-rewind modes. These modes could also be accomplished using a suitable order of I-frames.

Video information encoding is accomplished in the MPEG standard using DCT (discrete cosine transform). This technique represents wave form data as a weighted sum of cosines. DCT is also used for data compression in the JPEG standard.

Currently, there are several inadequate options from which to choose in order to make up for the lack of a high-performance multimedia toolkit. First, code could be developed from scratch as needed in order to solve a particular problem, but this is difficult given the complex multimedia standards such as JPEG and MPEG. Second, existing code could be modified but this results in systems that are complex, unmanageable, and generally difficult to maintain, debug, and reuse. Third, existing standard libraries like ooMPEG of the MPEG standard, or Independent JPEG Group (IJP) of the JPEG standard could be used, but the details of the functions in these libraries are hidden, and only limited optimizations can be performed.

It remains desirable to have a high-performance toolkit for multi-media processing.

It is an object of the present invention to provide a method and apparatus to enable client-server video-editing.

It is another object of the present invention to provide a method and apparatus to enable Web-based video-editing.

In accordance with the present invention. a network-based system is provided for processing multimedia information. A server, preferably a group of servers, are coupled to the network and each incorporates a multimedia toolkit or engine enabling creation, editing, viewing, and management of multimedia objects, such as files, including at least one of video, images, sound and animation. Each server includes storage for multimedia objects. A client, preferably a personal computer and preferably a plurality of clients, having access to the network each incorporate a multimedia-editing interface, preferably a graphic user interface (GUI), enabling the client to send multimedia processing commands through the network to the server from among a predefined set of multimedia processing commands. Preferably, a client can send a series of commands at a one time and each of several clients can send commands to be performed on the same object. The multimedia engine in each server acts on received multimedia processing commands from one or more clients by performing corresponding processing operations on multimedia objects previously stored by a client in the server's storage, and the server makes the processed multimedia objects available to clients over the network.

Preferably, the servers are controlled, as by a load-balancing daemon, so that clients are recognized and commands are routed to an appropriate server, each of the servers having access to storage containing the object to be processed, based on various criteria, including performance and feature availability. It is contemplated that authorized clients would be able to control such criteria.

The present invention together with the above and other advantages may best be understood from the from the following detailed description of the embodiments of the invention illustrated in the drawings, wherein:

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a Web-based video-editing system according to principles of the invention;

Figure 2 is a schematic of memory clipping according to the principles of the invention;

Figure 3 shows stereo samples interleaved in memory;

Figure 4 shows a toolkit function that performs the picture in picture operation according to principles of the invention;

Figure 5 shows the format of an MPEG-1 video stream;

Figure 6 shows a toolkit function that decodes the I-frames in an MPEG video into RGB images according to principles of the invention; and,

Figure 7 shows a toolkit function which acts as a filter that can be used to copy the packets of a first video stream from a system stream stored in a first BitStream to a second Bitstream according to principles of the invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Figure 1 shows a client/server Web-based video-editing system 10. A client computer (client) 15 is able to connect to a plurality of servers 20, 22, 24 over the Web 30. The client 15 runs a video-editing user interface 32. In the present embodiment, the Web 30 is used to connect the client 15 and the servers 20, 22, 24, however in alternative embodiments, other types of networks could be used to form the client-server connection. The servers 20, 22, 24 store audio and visual video data, and have multimedia processing applications.

In operation, the client 15 sends a command for video processing over the Web 30 to at least one of the servers 20, 22, 24. The at least one of the servers 20, 22, 24 receives the command, interprets it, performs the required video processing and sends the result to the client 15.

A more specific sequence of operation, might be:

The user sends a request from the client 15 to the server (A, B or C ※ 20, 21 or 24), establishing connection over the Web 30, between video editing user interface and video processing server with multimedia editing toolkit 34. The server, selected, for example, by a load-balancing daemon, recognizes the user as a client and awaits user commands. The user, utilizing video-editing user interface 32 ※ implemented, for example, in cross-platform Java or platform-based client, sends a command requesting processing of a certain operation on a specific media object stored at the server. The server verifies the command and locates the requested media object. The server locates an appropriate toolkit (34) which contains libraries for the requested operation and executes the operation, modifying the object. The server sends feedback to the client, for example via media streaming or other transfer protocol, informing of the executed operation and displaying the results in the user interface. This process is repeated until the user closes the connection.

The servers 20, 22, 24 have, as part of the video processing application, a high-performance toolkit (or

library) 34 according to the principles of the present invention. The present invention will be described in terms of the MPEG standard, however the principle of the invention may apply to other data standards. In addition, the MPEG standard is an evolving standard and the principles of the present invention may apply to MPEG standards yet to be developed.

In summary, Figure 1 describes an implementation of an online editing and browsing system. The Client-server application allows a user to edit, view and manage video as well as images, sound, animation and text over the network. This system is of a multi-component structure including one or more server(s) on the backend and one or more client(s). The system concentrates operations on the server side with commands being sent over the network from client to server. The basic idea of the client is to interact with a user. The client incorporates a video editing interface that allows the user to execute a set of predefined commands, pass them to the server and receive results for display. All or a majority of the Actual data processing occurs on the server, which provides the main data processing functionality and handles communication with remote clients.

The high-performance toolkit 34 provides code with performance competitive with hand-tuned C code, which allows optimizations to be performed without breaking open abstractions and is able to be composed by users in unforeseen ways. In order to accomplish high performance

multimedia data processing with predictable performance, resource control, and replacability and extensibility (i.e. usable in many applications), the present invention provides a toolkit, or API, was designed with the following properties.

The first property of the toolkit 34 is resource control. Resource control refers to control at the language level of I/O execution and memory allocation including reduction and/or elimination of unnecessary memory allocation. None of the toolkit routines of this invention implicitly allocate memory or perform I/O. The few primitives in the toolkit which do perform I/O, are primitives that load or store Bitstream data. The Bitstream is the actual stream of multimedia data. The MPEG bitstream will be discussed below. All other toolkit primitives of the invention use Bitstream as a data source. Users have full control over memory utilization and I/O. This feature gives users tight control over performance-critical resources, an essential feature for writing applications with predictable performance. The toolkit also gives users mechanisms to optimize programs using techniques such as data copy avoidance and to structure programs for good cache behavior.

The separation of I/O in the present invention has three advantages. First, it makes the I/O method used transparent to toolkit primitives. Generally, conventional libraries use integrated processing and I/O. A library that

integrates file I/O with its processing is difficult to use in a network environment, because the I/O behavior of networks is different from that of files. Second, the separation of I/O also allows control of when I/O is performed. It enables, for example, the building of a multithreaded implementation of the toolkit that allows the use of a double buffering scheme to read and process data concurrently. Third, by isolating the I/O calls, the performance of the remaining functions becomes more predictable.

The toolkit of this invention provides two mechanisms for sharing memory between abstractions, i.e. data objects. These mechanisms are called clipping and casting.

In clipping, one object "borrows" memory from another object of the same type. An example usage of clipping can be seen in Figure 2. In Figure 2, the abstraction ByteImage (a 2D array of values in the range 0 .. 255) is used to create a black box 60, then a gray box 62, and then the images are combined as a black box with a gray box inside 64 which share memory using the following pseudocode:

```
set a [byte_new 100 100]
      *** creates a new ByteImage of size 100 x 100 ***
byte_set $a 0
      *** initialize ByteImage to 0 (all black)  ***
set b [byte_clip $a 30 30 20 20]
     ***  creates a small ByteImage at position (30, 30)
          with dimensions of 20 x 20 inside $a.  Both a and
          b share the same memory ***
byte_set $b 128     *** initializes little box to gray  ***
byte_display $b     *** displays little gray box ***
byte_display $a
      *** displays black box with little gray box ***
```

Clipping functions are cheap in terms of memory usage because they allocate only an image header structure, and they are provided for in all of the toolkit's image and audio data types. Clipping is useful for avoiding unnecessary copying or processing of data. For example, if a user wants to decode only part of the gray scale image in an MPEG I-frame, the user could create a clipped DCTImage. A DCTImage is an array of DCT vectors, and a clipped DCTImage is an image that contains a subset of DCT blocks from the decoded I-frame. The user then performs an IDCT (inverse discrete cosine transform) on that clipped image to complete the decoding process. The advantage of this strategy is that it avoids performing the IDCT on encoded data that will not used.

Casting refers to sharing of memory between objects of different types. Casting is typically used for I/O, because all I/O is done through the BitStream. For instance, when a gray scale image file is read into the BitStream, the headers are parsed, and the remaining data is cast into a ByteImage. Casting avoids unnecessary copying of data.

In the toolkit, the user allocates and frees all non-trivial memory resources explicitly using new and free primitives, e.g., ByteImageNew and ByteImageFree. Functions never allocate temporary memory. If such memory is required to complete an operation (scra space, for example), the user must allocate it and pass it to the routine as a parameter. Explicit memory allocation allows the user

reduce or eliminate paging, and make the performance of the application more predictable.

For example, in the function ByteCopy, which copies one ByteImage to another, a potential problem is that the two ByteImages might overlap, e.g. if they share memory using clipping. A prior art method of implementing ByteCopy is:

```
(src, dest) {
malloc ();
src to temp;
temp to dest;
(temp);
```

```
ByteCopy
    temp =
    memcpy
    memcpy
    free
}
```

The implementation above allocates a temporary buffer, copies the source into the temporary buffer, copies the temporary buffers into the destination, and frees the temporary buffer. In contrast, the operation using the toolkit of the present invention is:

```
(src, dest) {
src to dest;

ByteNew ();
(src, temp);
(temp, dest);
(temp);
```

```
ByteCopy
    memcpy
}
temp =
ByteCopy
ByteCopy
ByteFree
```

The toolkit ByteCopy operation of the present invention assumes that the source and the destination do not overlap and it copies the source into the destination. The user must determine if the source and the destination overlap, and if they do, the user must allocate a temporary ByteImage and two ByteCopy calls as shown above.

The second property of the toolkit of this invention is that of having "thin" primitives. The toolkit breaks complex functions into simple functions that can be layered. This feature promotes code reuse and allows optimizations that would otherwise be difficult to exploit. For example, to decode a JPEG image, the toolkit provides three primitives: (1) a function to decode the bit stream into three DCTImages, one for each color component (the DCTImage is an array of DCT vectors), (2) a function to convert each DCTImage into a ByteImage (a simple image whose pixels are in the range 0..255), and (3) a function to convert from YUV color space to RGB color space.

Exposing this structure has several advantages. First, it promotes code reuse. For instance, the inverse DCT and color space conversion functions are shared by the JPEG and MPEG routines. Second, it allows optimizations that would be difficult to exploit otherwise. One such optimization is compressed domain processing. Another example is decoding a JPEG image to a gray scale image where only one DCTImage

needs to be decompressed, the DCTImage representing the gray scale component.

Many toolkit primitives of the present invention implement special cases of a more general operation. The special cases can be combined to achieve the same functionality of the general operation, and have a simple, fast implementation whose performance is predictable. ByteCopy is one such primitive - only the special case of non-overlapping images is implemented.

Another example is image scaling (shrinking or expanding the image). Instead of providing one primitive that scales an image by an arbitrary factor, the toolkit provides five primitives to shrink an image (Shrink4x4, Shrink2x2, Shrink2x1, Shrink1x2, and ShrinkBilinear), and five primitives to expand an image. Each primitive is highly optimized and performs a specific task. For example, Shrink2x2 is a specialized function that shrinks the image by a factor of 2 in each dimension. It is implemented by repeatedly adding 4 pixel values together and shifting the result, an extremely fast operation. Similar implementations are provided for Shrink4x4, Shrink2x1, and Shrink1x2. In contrast, the function ShrinkBilinear shrinks an image by a factor between 1 and 2 using bilinear interpolation. Although arbitrary scaling can be achieved by composing these primitives, splitting them into specialized operations makes the performance predictable, exposes the cost more

clearly to the user, and enables the user to produce very fast implementations.

The drawback to specialization in the present invention is that it can lead to an explosion in the number of functions in the API. Sometimes, however, the user can combine several primitives into one without sacrificing performance, which significantly reduces the number of primitives in the API. This principle is called generalization.

A good example of generalization is found in the primitives that process AudioBuffers. AudioBuffers store mono or stereo audio data. Stereo samples from the left and right channels are interleaved in memory as shown in Figure 3.

Suppose the user were implementing an operation that raises the volume on one channel (i.e., a balance control). One possible design is to provide one primitive that processes the left channel and another that processes the right channel as can be seen in the following code:

```
process_left
    for (i =
0, i < n, i +=2) {
        process x[i]
    }
    process_right
    for (i = 1, i < n, i +=2) {
    }.
```

The two operations, however, can combined without sacrificing performance by modifying the initialization of the looping variable (1 for right, 0 for left). This implementation is shown in the following code:

```
process (offset)
for (i = offset; i < n, i +=2) {
    process x[i]
}.
```

In general, if specialization gives better performance, it is recommended. Otherwise, generalization should be used to reduce the number of functions in the API.

The third property of the toolkit of the present invention is that of exposing structure. Most libraries try to hide details of encoding algorithms from the user, providing a simple, high-level API. In contrast, the present invention exposes the structure of compressed data in two ways.

First the toolkit exposes intermediate structures in the decoding process. For example, instead of decoding an MPEG frame directly into RGB format, the toolkit breaks the process into three steps: (1) bit stream decoding (including Huffman decoding and dequantization), (2) frame reconstruction (motion compensation and IDCT), and (3) color space conversion. For example, the MpegPicParseP function parses a P frame from a BitStream and writes the results into three DCTImages and one VectorImage. A second primitive reconstructs pixel data from DCTImage and

VectorImage data, and a third converts between color spaces. The important point is that the toolkit exposes the intermediate data structures, which allows the user to exploit optimizations that are normally impossible. For example, to decode gray scale data, one simply skips the frame reconstruction step on the Cr/Cb planes. Furthermore, compressed domain processing techniques can be applied on the DCTImage or VectorImage structures.

The toolkit of this invention also exposes the structure of the underlying bit stream. The toolkit provides operations to find structural elements in compressed bit streams, such as MPEG, JPEG and GIF. This feature allows users to exploit knowledge about the underlying bit stream structure for better performance. For example, a program that searches for an event in an MPEG video stream might cull the data set by examining only the I-frames initially, because they are easily (and quickly) parsed, and compressed domain techniques can be applied. This optimization can give several orders of magnitude improvement in performance over conventional event-searching methods in some circumstances, but because other libraries hide the structure of the MPEG bit stream from the user, this optimization cannot be used. In the present invention, this optimization is trivial to exploit. The user can use the MpegPicHdrFind function to find a picture header, MpegPicHdrParse to decode it, and, if the *type* field in the

decoded header indicates the picture that follows is an I-frame, can use MpegIPicParse to decode the picture.

The toolkit provides a plurality of basic abstractions. These basic abstractions are:

- ByteImage ※ a 2D array of values in the range 0..255.
- BitImage ※ a 2D array of 0/1 values.
- DCTImage ※ a 2D array of elements, each of which is a sequence of (*index*,*value*) pairs representing the run-length-encoded DCT blocks found in many block-based compression schemes, such as MPEG and JPEG.
- VectorImage ※ a 2D array of vectors, each with a horizontal and vertical component, representing motion-vectors found in MPEG or H.261.
- AudioBuffer ※ a 1D array of 8 or 16-bit values.
- ByteLUT ※ a look-up table for ByteImages. A ByteLUT can be applied to one ByteImage to produce another ByteImage.
- AudioLUT ※ a look-up tables for AudioBuffers.
- BitStream/BitParser ※ A BitStream is a buffer for encoded data. A BitParser provides a cursor into the BitStream and functions for reading/writing bits from/to the BitStream.
- Kernel ※ 2D array of integers, used for convolution.

- Filter - a scatter/gather list that can be used to select a subset of a BitStream.

These abstractions can be used to represent common multimedia data objects. For example,

- A gray-scale image can be represented using a ByteImage.
- A monochrome image can be represented using a BitImage.
- An irregularly shaped region can be represented using a BitImage.
- An RGB image can be represented using three ByteImages, all of the same size.
- An YUV image in 4:2:0 format can be represented using three ByteImages. The ByteImage that represents the Y plane is twice the width and height of the ByteImages that represent the U and V planes.
- The DCT blocks in a JPEG image, an MPEG I-frame, or the error terms in an MPEG P- and B-frame can be represented using three DCTImages, one for each of the Y, U and V planes of the image in the DCT domain.
- The motion vectors in MPEG P- and B-frame can be represented with a VectorImage.
- A GIF Image can be represented using three ByteLUTs, one for each color map, and one ByteImage for the color-mapped pixel data.

- 8 or 16-bit PCM audio, 16-bit PCM audio, μ-law or A-law audio data can be represented using an AudioBuffer. The audio can be either stored as single channel or contain both left and right channels.

The toolkit also has abstractions to store encoding-specific information. For example, an MpegPicHdr stores the information parsed from a picture header in an MPEG-1 video bit stream. The full list of header abstractions can be found in Table 1.

Header	File Format
PnmHdr	NETPBM image header
WavHdr	WAVE audio header
GifSeqHdr	GIF file sequence header
GifImgHdr	GIF file image header
JpegHdr	JPEG image header
JpegScanHdr	JPEG scan header
MpegAudioHdr	MPEG-1 audio (layer 1, 2, 3) header
MpegSeqHdr	MPEG-1 video sequence header
MpegGopHdr	MPEG-1 video group-of-picture header
MpegPicHdr	MPEG-1 video picture header
MpegSysHdr	MPEG-1 system stream system header
MpegPckHdr	MPEG-1 system stream pack header
MpegPktHdr	MPEG-1 system stream packet header

Table 1. Header abstractions

Although the set of abstractions defined in the toolkit is fairly small, the set of operators that manipulate these abstractions is not.

The following examples, relating to the present invention, illustrate the use of the abstractions in the

toolkit and demonstrate writing programs using the toolkit. The first example shows how to use the toolkit to manipulate images. The second example shows how to use the toolkit's primitives and abstractions for MPEG decoding. The third example shows how to use a toolkit filter to demultiplex an MPEG systems stream.

The first example is a simple example of using the toolkit to manipulate images. The ByteImage function will be used. A ByteImage consists of a *header* and a *body*. The header stores information such as width and height of the ByteImage and a pointer to the body. The body is a block of memory that contains the image data. A ByteImage can be either *physical* or *virtual*. The body of a physical ByteImage is contiguous in memory, whereas a virtual ByteImage borrows its body from part of another ByteImage (called its *parent*). In other words, a virtual ByteImage provides a form of shared memory ※ changing the body of a virtual ByteImage implicitly changes the body of its parent, as seen in Figure 2.

A physical ByteImage is created using ByteNew(w,h). A virtual ByteImage is created using ByteClip(b, x, y, w, h). The rectangular area whose size is w x h and has its top left corner at (x,y) is shared between the virtual ByteImage and the physical ByteImage. The virtual/physical distinction applies to all image types in the toolkit. For example, a virtual DCTImage can be created to decode a subset of a JPEG image.

In an operation of creating a "picture in picture" (PIP) effect on an image, the steps creating the PIP effect are as follows: Given an input image (1) scale the image by half, (2) draw a white box slightly larger than the scaled image on the original image, and (3) paste the scaled image into the white box.

The code in Figure 4 shows a toolkit function that performs the PIP operation. The function takes in three arguments: image, the input image; borderWidth, the width of the border around the inner image in the output, and margin, the offset of the inner image from the right and bottom edge of the outer image.

Line 5 to line 6 of the function query the width and height of the input image. Line 7 to line 10 calculate the position and dimension of the inner picture. Line 13 creates a new physical ByteImage, temp, which is half the size of the original image. Line 14 shrinks the input image into temp. Line 15 creates a virtual ByteImage slightly larger than the inner picture, and line 18 sets the value of the virtual ByteImage to 255, achieving the effect of drawing a white box. Line 19 de-allocates this virtual image. Line 20 creates another virtual ByteImage, corresponding to the inner picture. Line 21 copies the scaled image into the inner picture using ByteCopy. Finally lines 22 and 23 free the memory allocated for the ByteImages.

This example shows how images are manipulated in the toolkit through a series of simple, thin operations. It

also illustrates several design principles of the toolkit, namely (1) sharing of memory (through virtual images), (2) explicit memory control (through ByteClip, ByteNew and ByteFree), and (3) specialized operators (ByteShrink2x2).

The second example relating to this invention illustrates how to process MPEG video streams using the toolkit. The example program decodes the I-frames in an MPEG video stream into a series of RGB images. To parse an MPEG video stream, the encoded video data is first read into a BitStream. A BitStream is a abstraction for input/output operations - that is, it is a buffer. To read and write from the BitStream, a BitParser is used. A BitParser provides functions to read and write data to and from the BitStream, plus a cursor into the BitStream.

Figure 5 shows the format of an MPEG-1 video stream 150. The MPEG video stream 150 has of a *sequence header* 152, followed by a sequence of *GOPs (group-of-pictures)* 154, followed by an *end of sequence marker* 156. Each GOP consists of a GOP header 158 followed by a sequence of *pictures* 160. Each picture consists of a *picture header* 162, followed by a picture body 164 which is made up of compressed data required to reconstruct the picture. The sequence header 152 contains information such as the width and height of the video, the frame rate, the aspect ratio, and so on. The GOP header 158 contains the timecode for the GOP. The picture header 162 contains information necessary for decoding the picture, most notably the type of picture

(I, P, B). The toolkit provides an abstraction for each of these structural elements (see Table 1).

The toolkit provides five primitives for each structural element: find, skip, dump, parse, and encode. Find positions the cursor in the BitStream just before the element. Skip advances the cursor to the end of the element. Dump moves the bytes corresponding to the element from input BitStream to the output BitStream, until the cursor is at the end of the header. Parse decodes the BitStream and stores the information into a header abstraction, and encode encodes the information from a header abstraction into a BitStream. Thus, the MpegPicHdrFind function advances the cursor to the next picture header, and MpegSeqHdrParse decodes the sequence header into a structure.

These primitives provide the necessary functions to find, skip, or parse MPEG I-frames. The parsed picture data from an MPEG I-frame is represented using a DCTImage. A DCTImage is similar to ByteImage, but each "pixel" is an 8x8 DCT encoded block.

The toolkit code in Figure 6 decodes the I-frames in an MPEG video into RGB images. Lines 1 through 5 allocate the data structures needed for decoding. Line 6 attaches the BitParser inbp to BitStream inbs. The cursor of inbp will be pointing to the first byte of buffer in inbs. Line 7 fills inbs with 64K of data from the input MPEG video. Line 8 moves the cursor of inbp to the beginning of a sequence

header, and line 9 parses the sequence header and stores the information from the sequence header into the structure seqhdr.

The vital information such as width, height and the minimum data that must be present to decode a picture (vbvsize) is extracted from the sequence header in lines 10 through 12. Lines 13 through 21 allocate the ByteImages and DCTImages we need for decoding the I-frames. Y, u, and v store the decoded picture in YUV color space, and r, g, and b store the decoded picture in RGB color space. Dcty, dctu, and dctv store compressed (DCT domain) picture data. The main loop in the decoding program (lines 22-46) starts by advancing the BitParser cursor to the beginning of the next marker (line 24) and retrieves the current marker (line 25).

If the marker indicates the beginning of a picture header, the picture header is parsed (line 28) and its type is checked (line 29). If the picture is an I-frame, the I-frame is parsed into three DCTImages, (line 30), the DCTImages are converted to ByteImages (lines 31-33), and the ByteImages are converted into RGB color space (line 34).

If the marker indicates the beginning of a GOP header, the header is skipped (which moves the cursor to the end of the GOP header), because information from the GOP header is not needed.

Finally, if the sequence end marker is encountered, that marks the end of the video stream and the loop is exited. Lines 43-45 ensure that, during the next iteration

of the main loop, inbs will contain sufficient data to continue decoding. UpdateIfUnderflow checks if the number of bytes remaining in a inbs is less than vbvsize. If so, the remaining bytes are shifted to the beginning of the buffer and the rest of the buffer filled with data from file.

Breaking down complex decoding operations like MPEG decoding into " thin" primitives makes the toolkit code highly configurable. For example, by removing lines 32 to 34, the program decodes MPEG I-frame into gray scale images. By replacing line 31 to 34 with JPEG encoding primitives, an efficient MPEG I-frame to JPEG transcoder is produced.

The third example relating to this invention illustrates filtering out a subset of a BitStream for processing. Filters were designed to simplify the processing of bit streams with interleaved data (e.g., AVI, QuickTime, or MPEG systems streams). Filters are similar to scatter/gather vectors - they specify an ordered subset of a larger set of data.

A common use of filtering is processing MPEG system streams, which includes interleaved audio or video (*A/V*) streams. In MPEG, each A/V stream is assigned an unique id. Audio streams have ids in the range 0..31; video streams ids are in the range 32..47. The A/V streams are divided up into small (approx. 2 KByte) chunks, called *packets*. Each packet has a header that contains the id of the stream, the

length of the packet, and other information (e.g., a timecode).

In this example, a filter is built that can be used to copy the packets of the first video stream (id = 32) from a system stream stored in one BitStream to another. Once copied, the toolkit MPEG video processing primitives can be used on the video-only BitStream. The toolkit code for building this filter is shown in Figure 7.

Lines 2 through 8 allocate and initialize various structures needed by this program. The variable offset stores the byte offset of a packet in the bit stream, relative to the start of the stream. Line 9 advances the cursor to the beginning of the first packet header and updates offset. The main loop (lines 10-18) parses the packet header (line 11) and, if the packet belongs to the first video stream, its offset and length are added to filter (line 14). EndOfBitstream is a macro that checks the position of the bit stream cursor against the length of the data buffer.

Once the filter is constructed, it can be saved to disk, used as a parameter to the BitStreamFileFilter or the BitstreamDumpUsingFilter functions. The former reads the subset of a file specified by the filter, the latter copies the data subset specified by a filter from one bit stream to another.

This example illustrates how the toolkit can be used to demultiplex interleaved data. It can be easily extended to

other formats, such as QuickTime, AVI, MPEG-2 and MPEG-4. Although this mechanism uses data copies, the cost of copying is offset by the performance gain when processing the filtered data.

It is to be understood that the above-described embodiments are simply illustrative of the principles of the invention. Various and other modifications and changes may be made by those skilled in the art which will embody the principles of the invention and fall within the spirit and scope thereof.

WE CLAIM:

1. A network-based system for processing multimedia information, comprising:

a server module coupled to the network and incorporating a multimedia engine enabling creation, editing, viewing, and management of multimedia objects including at least one of video, images, sound and animation, the server having access to storage for multimedia objects;

a client module having access to the network and incorporating a multimedia-editing interface, enabling the client to send multimedia processing commands through the network to the server from among a predefined set of multimedia processing commands, the multimedia engine in the server acting on received multimedia processing commands from the client by performing corresponding processing operations on multimedia objects previously stored by the client in the server's storage, the server making the processed multimedia objects available to the client.

2. A system in accordance with claim 1, further comprising an additional server module coupled to the network and incorporating a multimedia engine enabling editing, viewing, and management of multimedia objects including at least one of images, sound and animation, the server having access to storage for multimedia objects, and a control module controlling access to said server modules by a client module on the basis of predefined criteria, the multimedia engine in at least one of said servers acting on received multimedia processing commands from the client by performing corresponding processing operations on multimedia objects previously stored, the servers making the processed multimedia objects available to the client.

3. A system in accordance with claim 2 wherein said control module is a load-balancing daemon which recognizes the client, awaits commands, and passes them to a selected server based on the criteria.

4. A system in accordance with claim 3 wherein said control module is

configured to permit an authorized client to establish and modify the criteria.

5. A system in accordance with claim 3, further comprising an additional client module having access to the network and incorporating a multimedia-editing interface, enabling the client to send multimedia processing commands through the network to at least one of the servers, the processing commands being from among a predefined set of multimedia processing commands.

6. A system in accordance with claim 5 wherein said control module is configured to permit more than one client at time to submit multimedia processing commands related to a particular object.

7. A system in accordance with claim 5 wherein said control module is configured to permit interaction among clients.

8. A system in accordance with claim 1 or 2, further comprising an additional client module having access to the network and incorporating a multimedia-editing interface, enabling the client to send multimedia processing commands through the network to at least one of the servers, the processing commands being from among a predefined set of multimedia processing commands.

9. A system in accordance with claim 1 or 2, wherein a server provides processed multimedia objects in the form of streaming media..

10. A method for providing network-based processing of multimedia information, comprising the steps of:

at a first location providing a server computer running a multimedia engine program enabling creation, editing, viewing, and management of multimedia objects including at least one of video, images, sound and animation, the server having access to storage for

multimedia objects, as well as a connection to a network;

at a second location, remote from the first location, providing a client computer having access to the network and running a multimedia-editing interface program enabling the client to send multimedia processing commands through the network to the server computer from among a predefined set of multimedia processing commands;

at the first location, causing the multimedia engine program to act on received multimedia processing commands from the client by performing corresponding processing operations on multimedia objects previously stored in the server's storage, and making the processed multimedia objects available to the client computer over the network.

11. The method of claim 10, further comprising:

providing an additional server computer running a multimedia engine program enabling editing, viewing, and management of multimedia objects including at least one of images, sound and animation, and additional storage for multimedia objects, as well as a connection to a network; and

controlling access to said server computers by a client computer on the basis of predefined criteria;

the multimedia engine program in at least one of said servers acting on received multimedia processing commands from the client by performing corresponding processing operations on multimedia objects previously stored by the client in said at least one server's storage;

making available to the client computer over the network multimedia objects from at least one of the servers.

12. The method of claim 11 wherein said controlling step comprises load-balancing, recognizing the client, awaiting commands, and passing them to a selected server based on the criteria.

13. The method of claim 12 wherein said controlling step comprises permitting

an authorized client to establish and modify the criteria.

14. The method of claim 13, further comprising:

providing an additional client computer having access to the network and incorporating a multimedia-editing interface, enabling the client to send multimedia processing commands through the network to at least one of the server computers, the processing commands being from among a predefined set of multimedia processing commands.

14. The method of claim 13 wherein said controlling step includes permitting more than one client at time to submit multimedia processing commands related to a particular object.

15. The method of claim 13 wherein said controlling step includes permitting interaction among clients.

16. The method of claim 10 or 11, further comprising:

providing an additional client computer having access to the network and incorporating a multimedia-editing interface, enabling the client to send multimedia processing commands through the network to at least one of the server computers, the processing commands being from among a predefined set of multimedia processing commands.

17. The system of any of claims 1-9 wherein the multimedia editing interface is a graphic ser interface.

18. The system of any of claims 1-9 wherein the multimedia engine accepts a plurality of commands from a client at one time.

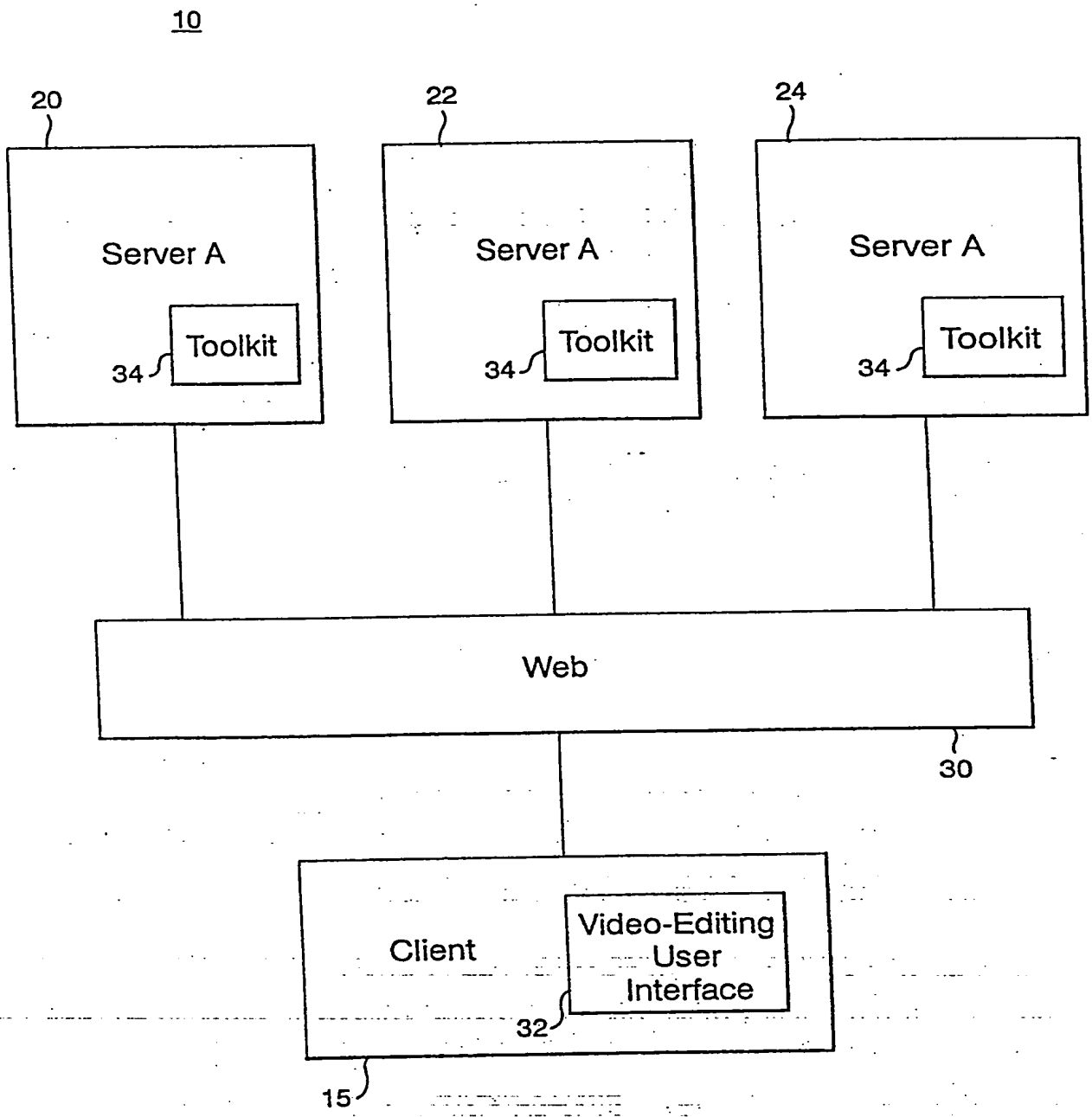

10
20
22
24
Server A
Toolkit
34
Server A
Toolkit
34
Server A
Toolkit
34
Web
30
Client
Video-Editing User Interface
32
15

Fig. 1

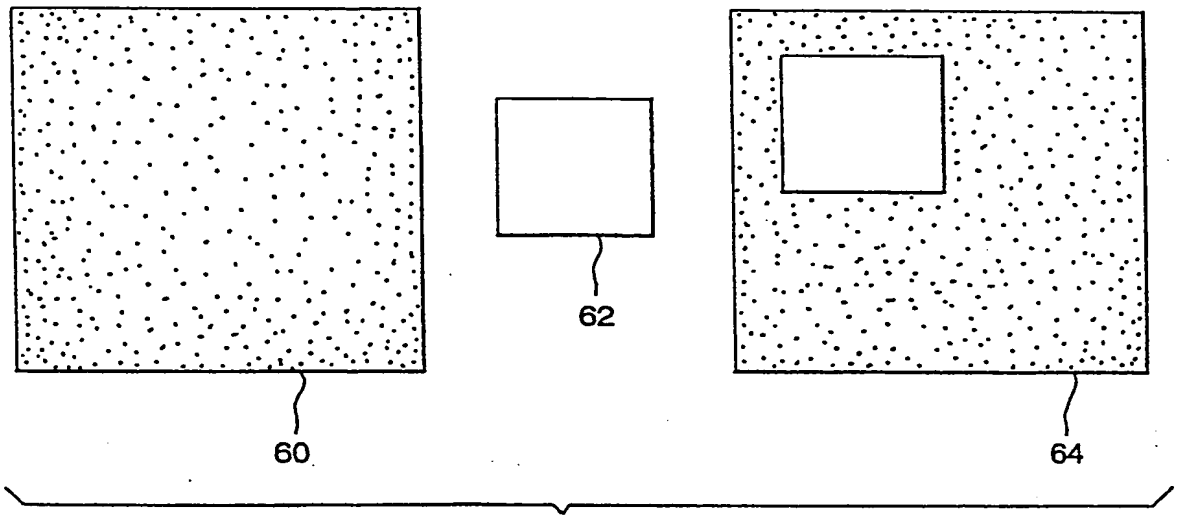


Fig. 2

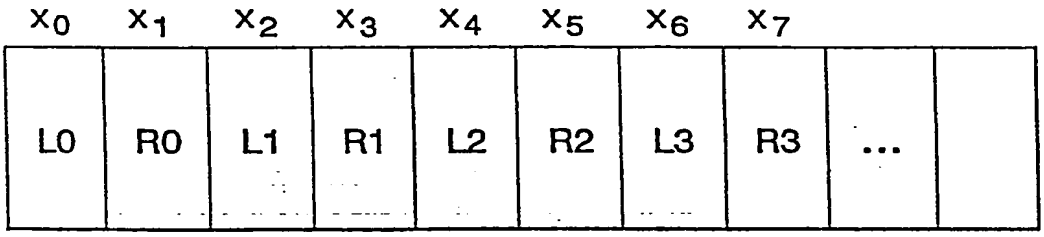


Fig. 3

```
1     Void PIP(image, borderWidth, margin)
2      ByteImage *image;
3      int borderWidth, margin;
4     (
5         int w   = ByteGetWidth(image);
6         int h   = ByteGetHeight(image);
7         int destW = w/2;
8         int destH = h/2;
9         int destX = w - destW - margin;
10        int destY = h - destH - margin;
11        ByteImage *dest;
12        ByteImage *temp;

13    temp = ByteNew(destW,destH);
14    ByteShrink2x2 (image, temp);

15       dest = ByteClip(image,
16         destX-borderWidth, destY-borderWidth,
17         destW+2*borderWidth,destH+2*borderWidth);
18       ByteSet(dest, 255);
19       ByteFree(dest);

20       dest = ByteClip(image,
21         destX, destY, destW, destH);
22       ByteCopy(temp, dest);
23    ByteFree(dest);
24    ByteFree(temp);
25    )
```

Fig. 4

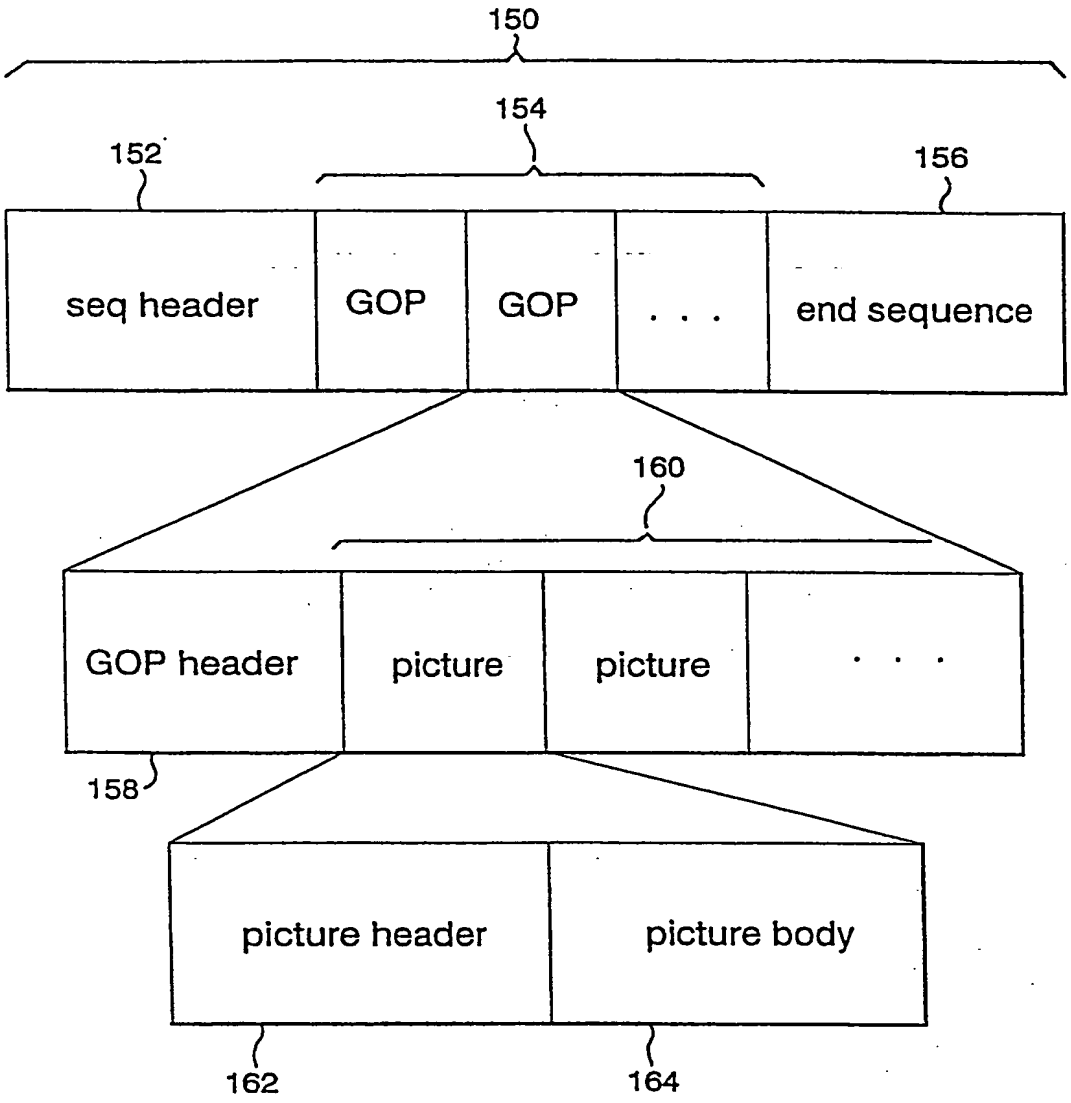

150
154
152
156
seq header
GOP
GOP
. . .
end sequence
160
GOP header
picture
picture
. . .
158
picture header
picture body
162
164

Fig. 5

```
//  file is the input sources for the MPEG streams
1   BitStream      *inbs = BiStreamNew (65536);
2   BitParser      *inbp = BitParserNew ();
3   MpegSeqHdr     *seqhdr = MpegSeqHdrNew ();
4   MpegPicHdr *pichdr = MpegPicHdrNew ();
5   int W, h, vbvsize, done=0;
6   BitParserAttach (inbp, inbs);
7   BitStreamFileRead (inbs, file, 0);
8   MpegSeqHdrFind (inbp)
9   MpegSeqHdrParse (inbp, seqhdr);
10  w = seqhdr->width;
11  h = seqhdr->height;
12  vbvsize = seqhdr->vbv_buf_size;
13  r = ByteNew (w, h);
14  g = ByteNew (w, h);
15  b = ByteNew (w, h);
16  y = ByteNew (w, h);
17  u = ByteNew(w/2, h/2);
18  v = ByteNew(w/2, h/2);
19  dcty = DctNew(w/16, h/16);
20  dctu = DctNew(w/32, h/32);
21  dctv = DctNew(w/32, h/32);
22  while (!done) {
23      int marker;
24      mpeg_any_markerFind (inbp);
25      marker = MpegGetCurrMarker (inbp);
26      switch (market) {
27       case PIC_HDR_MARKER :
28         MpegPicHdrParse (inbp, pichdr);
29         if (pichdr->type == I_FRAME)  {
30           MpegPicIParse (inbp,dcty,dctu,dctv);
31           DctToByte (dcty, y);
32           DctToByte (dctu, u);
33           DctToByte (dctv, v);
34           YuvToRgb420 (y, u, v, r, g, b);
35         )
36         break;
37       case COP_HDR_MARKER :
38         MpegGopHdrSkip (inbp);
39         break;
40       case SEQ_END_MARKER :
41         done = 1;
42      )
43  if (!done)  {
44      UpdateIfUnderflow (inbp,inbs,file,vbvsize);
45  }
46  }
```

Fig. 6

```
1    #define SIZE (128*1024)
2    int len, offset, start = 0;
3    MpegPktHdr *hdr = MpegPktHdrNew();
4    BitStream *bs = BitStreamNew (SIZE);
5    BitParser *bp = BitParserNew ();
6    BitStreamFilter * filter = BitStreamFilterNew();

7    BitParserAttach (bp, bs);
8    BiStreamFileRead (bs, file);

9    offset = MpegPktHdrFind (bp);
10   while (!eof(file) && !EndOfBitstream(bp))   {
11     MpegPktHdrParse (bp, hdr);
12     if (hdr->id == 32)  {
13       len = hdr->len;
14       BitStreamFilterAdd(filter, offset, len);
15       start += UpdateIfUnderflow (bp,bs,file,SIZE);
16       offset = start + MpegBktHdrFind(bp);
17     }
18   }
```

Fig. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/29961

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :G06F 13/00, 3/00; G06T 13/00.
US CL :345/335, 328.
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 345/335, 328, 327, 333, 334.

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searche

none

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used

APS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim
X	US 5,559,949 A (REIMER ET AL.) 24 SEPTEMBER 1996, COL. 4, LINE 66, TO COL. 22, LINE 56.	1-18
A	US 5,740,388 A (HUNT) 14 APRIL 1998, THE WHOLE DOCUMENT.	1-18

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 29 FEBRUARY 2000	Date of mailing of the international search report 21 MAR 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer MICHAEL LEE James R. Matthews Telephone No. (703) 305-7000

Form PCT/ISA/210 (second sheet)(July 1992)*

This Page Blank (uspto)

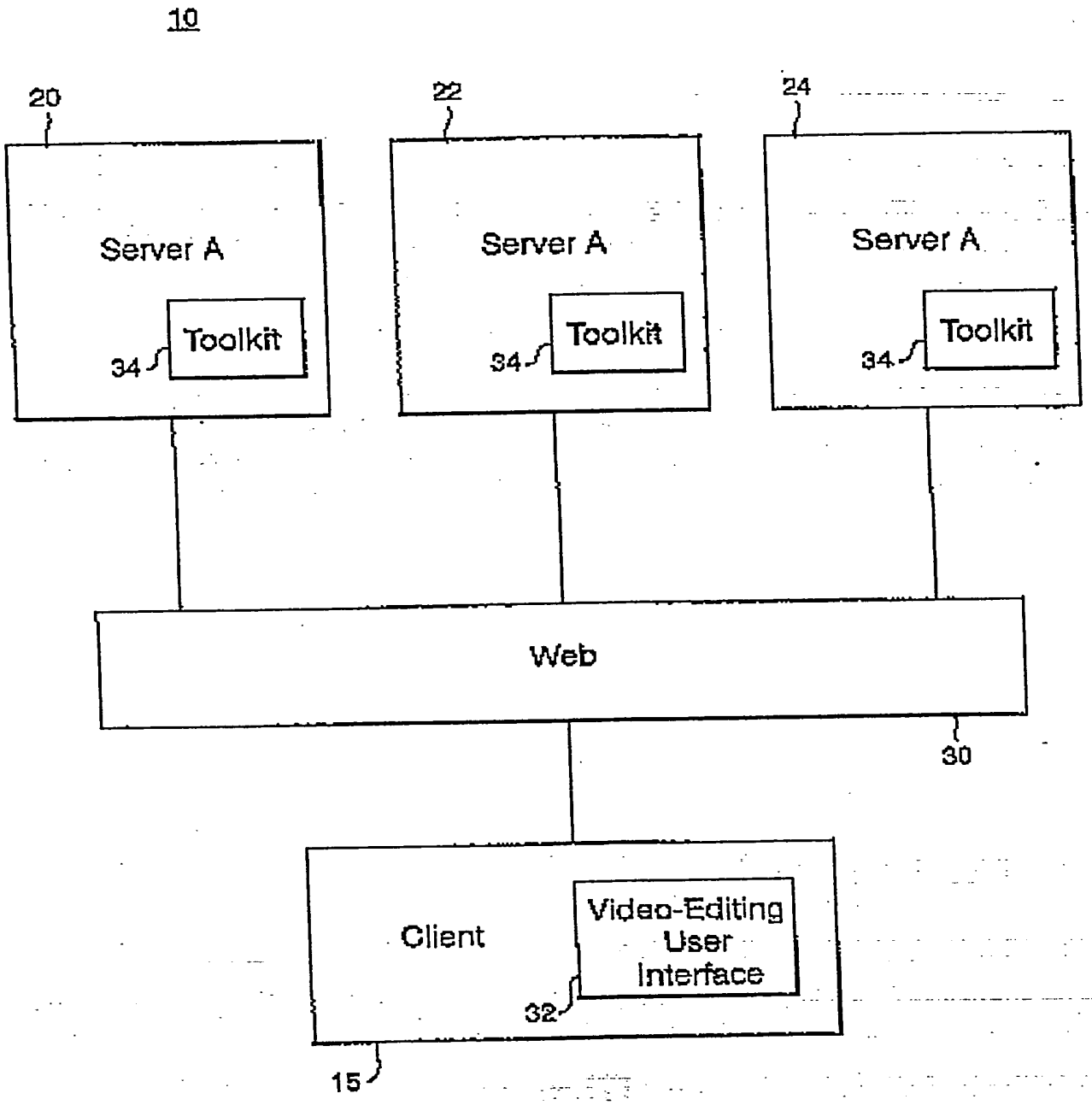

10
20
22
24
Server A
Toolkit
34
Server A
Toolkit
34
Server A
Toolkit
34
Web
30
Client
Video-Editing User Interface
32
15

Fig. 1

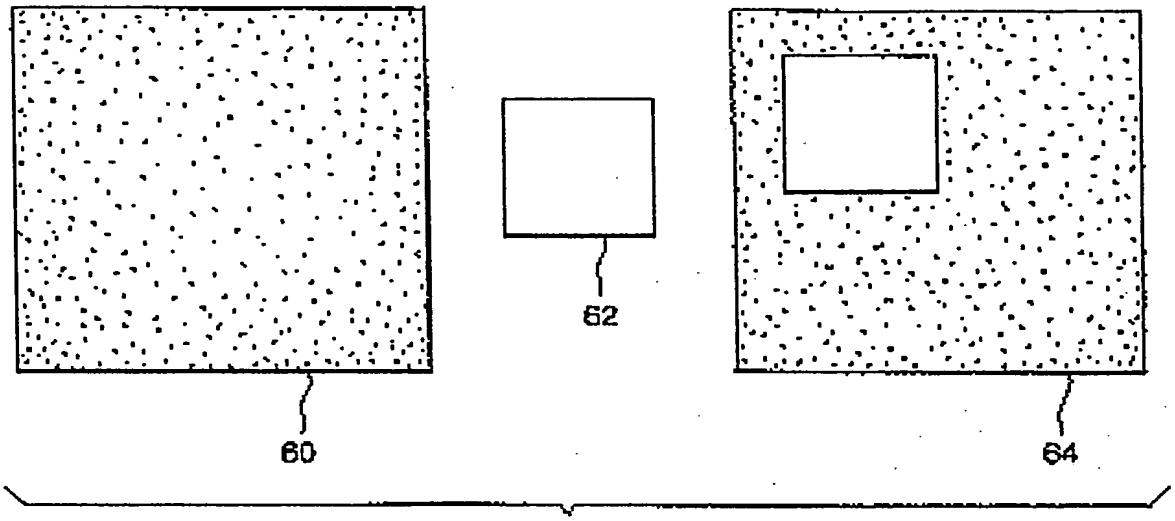


Fig. 2

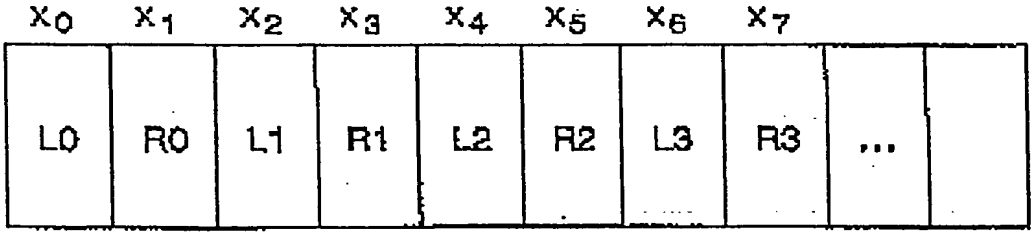


Fig. 3

```
1    Void PIP(image, borderWidth, margin)
2     ByteImage *image;
3     int borderWidth, margin;
4    (
5        int w   = ByteGetWidth(image);
6        int h   = ByteGetHeight(image);
7        int destW = w/2;
8        int destH = h/2;
9        int destX = w - destW - margin;
10       int destY = h - destH - margin;
11       ByteImage *dest;
12       ByteImage *temp;

13   temp = ByteNew(destW,destH);
14   ByteShrink2x2 (image, temp);

15      dest = ByteClip(image,
16        destX-borderWidth, destY-borderWidth,
17        destW+2*borderWidth,destH+2*borderWidth);
18      ByteSet(dest, 255);
19      ByteFree(dest);

20      dest = ByteClip(image,
21        destX, destY, destW, destH);
22      ByteCopy(temp, dest);
23   ByteFree(dest);
24   ByteFree(temp);
25   )
```

Fig. 4

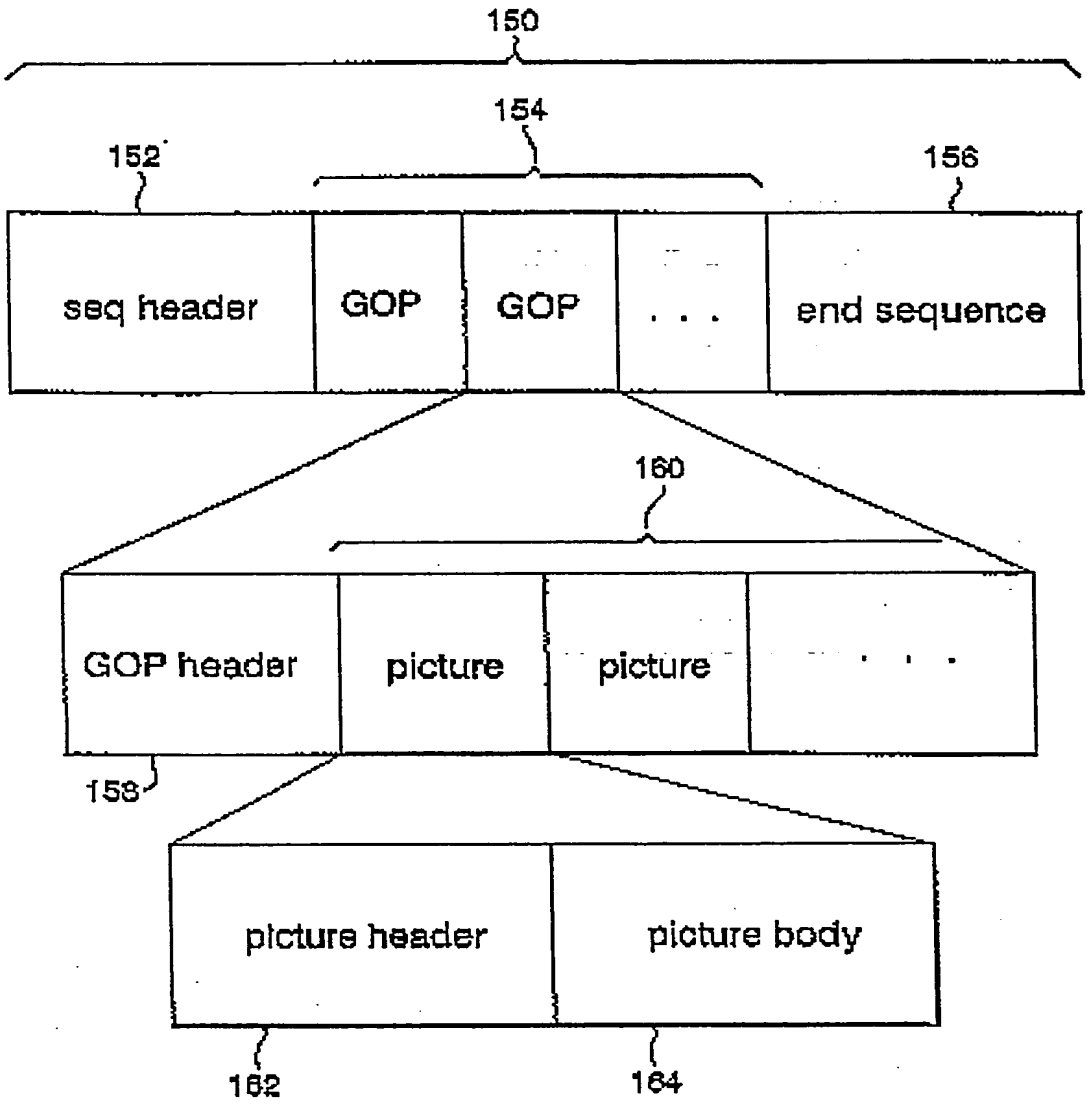

150
154
152
156
seq header
GOP
GOP
. . .
end sequence
160
GOP header
picture
picture
. . .
158
picture header
picture body
162
164

Fig. 5

```
//  file is the input sources for the MPEG streams
1   BitStream      *inbs = BiStreamNew (65536);
2   BitParser      *inbp = BitParserNew ();
3   MpegSeqHdr     *seqhdr = MpegSeqHdrNew ();
4   MpegPicHdr *pichdr = MpegPicHdrNew ();
5   int W, h, vbvsize, done=0;
6   BitParserAttach (inbp, inbs);
7   BitStreamFileRead (inbs, file, 0);
8   MpegSeqHdrFind (inbp)
9   MpegSeqHdrParse (inbp, seqhdr);
10  w = seqhdr->width;
11  h = seqhdr->height;
12  vbvsize = seqhdr->vbv_buf_size;
13  r = ByteNew (w, h);
14  g = ByteNew (w, h);
15  b = ByteNew (w, h);
16  y = ByteNew (w, h);
17  u = ByteNew(w/2, h/2);
18  v = ByteNew(w/2, h/2);
19  dcty = DctNew(w/16, h/16);
20  dctu = DctNew(w/32, h/32);
21  dctv = DctNew(w/32, h/32);
22  while (!done) {
23      int marker;
24      mpeg_any_markerFind (inbp);
25      marker = MpegGetCurrMarker (inbp);
26      switch (market) {
27       case PIC_HDR_MARKER :
28         MpegPicHdrParse (inbp, pichdr);
29         if (pichdr->type == I_FRAME)  {
30           MpegPicIParse (inbp,dcty,dctu,dctv);
31           DctToByte (dcty, y);
32           DctToByte (dctu, u);
33           DctToByte (dctv, v);
34           YuvToRgb420 (y, u, v, r, g, b);
35         }
36         break;
37       case GOP_HDR_MARKER :
38         MpegGopHdrSkip (inbp);
39         break;
40       case SEQ_END_MARKER :
41         done = 1;
42      }
43  if (!done)  {
44      UpdateIfUnderflow (inbp,inbs,file,vbvsize);
45  }
46  }
```

Fig. 6

```
1   #define SIZE (128*1024)
2   int len, offset, start = 0;
3   MpegPktHdr *hdr = MpegPktHdrNew();
4   BitStream *bs = BitStreamNew (SIZE);
5   BitParser *bp = BitParserNew ();
6   BitStreamFilter * filter = BitStreamFilterNew();

7   BitParserAttach (bp, bs);
8   BitStreamFileRead (bs, file);

9   offset = MpegPktHdrFind (bp);
10  while (!eof(file) && !EndOfBitstream(bp))  {
11    MpegPktHdrParse (bp, hdr);
12    if (hdr->id == 32)  {
13      len = hdr->len;
14      BitStreamFilterAdd(filter, offset, len);
15      start += UpdateIfUnderflow (bp,bs,file,SIZE);
16      offset = start + MpegPktHdrFind(bp);
17    }
18  }
```

Fig. 7